USING C WRAPPERS IN XLISP-STAT

JAN DE LEEUW

Abstract. This paper discusses incorporating C (and FORTRAN) functions into the XLISP-STAT statistical computing environment, by using shared libraries loaded at runtime. We provide a number of examples that can be used as templates. They can be used (as in S-plus) to speed up XLISP-STAT programs, but also (as in SWIG) to produce graphical user interfaces for existing C and FORTRAN programs. The appendices discuss a number of completed projects following these lines which considerably extend XLISP-STAT.

Contents
1. Introduction 2
2. Lisp Code, Byte Code, Object Code 3
3. Shared Libraries: Example 1 5
4. Shared Libraries: Example 2 7
5. Shared Libraries: Example 3 9
Appendix A. Introduction 11
Appendix B. cephes 12
Appendix C. gd 19
Appendix D. pppack 25
Appendix E. specfun 26
Appendix F. probability 29
Appendix G. smoothing 30
Appendix H. solving 31
Appendix I. optimization 32
Appendix J. transform 33
Appendix K. density 34
References 35

Date: June 24, 1999.
We thank Luke Tierney for help along the way.
1. Introduction

XLISP-STAT [Tierney, 1990] can be extended in two different ways. By far the most common one is to write additional code in Lisp, which is read into the interpreter at run-time. Much less common is to extend the system by writing additional code in C.

There are basically two reasons to use C. In the first place, we may have legacy C code, for instance from books such as Press et al. [1988], and we may not have the time or the resources to translate this C to Lisp. Secondly, we may have a project, or a part of a project, which needs to run very fast, faster than is possible in the interpreted Lisp environment of XLISP-STAT. In the interpreted environment we can do fast prototyping, and we can write elaborate and elegant graphical interfaces. Fast floating point computation, however, is only possible if the necessary functions are already linked as object code into the XLISP-STAT executable.

There are two ways to incorporate C in XLISP-STAT. It can be done at compile-time, when the XLISP-STAT executable is build. One simply writes additional functions in C, and one uses the XLISP-STAT application programmer interface to make them accessible to the interpreter. This is not hard to do, but it has a major disadvantage. Anybody who does this will have created a personal copy of XLISP-STAT, different from all other copies in the world. It will be difficult to maintain and to upgrade, unless the extensions are incorporated (by Luke Tierney) in the canonical XLISP-STAT source tree. Extending at compile-time also means you need the tools to build a complete XLISP-STAT distribution. On the Mac, for instance, this implies you need to have the Metrowerks CodeWarrior Pro tools, because that is the only supported development environment.

Alternatively, it is possible to load precompiled C code at run-time. This is what we discuss in this note. It requires you to have utilities for building shared libraries, but this can be done with many different sets of tools. On the Mac, for instance, the free MPW and GNU environments, with the MrC and gcc compilers, can be used.

The Lisp tools and XLISP-STAT “wrapper” extensions to build these shared libraries have been developed by Luke Tierney. The technical aspects and the implementation are discussed in detail by Tierney [1998c]. Background information on dynamic loading, native pointers, and shared libraries is in Tierney [1998b,d,f]. Applications that create a regular expression library and a socket interface for XLISP-STAT are in Tierney [1998e,g].
In this paper we strip away as much of the technical detail as possible, and concentrate on simple computational examples. For a real understanding of the implementation on the various operating systems, we refer to Tierney’s papers. They can all be found at the URL

www.stat.umn.edu/~luke/xls/projects/

2. Lisp Code, Byte Code, Object Code

We can be a little bit more specific here. Let us take the inner product function as an example. The inner product is defined in the file linalg.lsp in the XLISP-STAT distribution. That file gets byte-compiled during installation, and normally it sits as a byte-compiled function in the XLISP-STAT workspace. We can show this by looking in the function slot of the inner-product function.

> (symbol-function 'inner-product)
#<Byte-Code-Closure-INNER-PRODUCT: #54ae758>

Byte-compilation transforms Lisp code into instructions for the XLISP-STAT virtual machine, which runs on the various platforms that XLISP-STAT has been ported to. Usually, byte-compiled code is considerably faster than interpreted Lisp code. But much less fast than the native object code for the specific processor that regular C or FORTRAN compilers produce.

Fortunately, the inner-product function is just a high-end interface to the blas-ddot function. It does some error testing in Lisp, and then calls blas-ddot to do the work. And if we look in the corresponding function slot, we see

> (symbol-function 'blas-ddot)
#<Subr-BLAS-DDOT: #548aa18>

Thus blas-ddot is a subr, which means a compiled function living in object code in the XLISP-STAT executable. It is taken from the C version of the BLAS Anderson et al. [1992], which is a library of highly efficient building blocks for numerical linear algebra. In the XLISP-STAT distribution it is in the file blas.c. Thus the critical parts of inner-product, which is where the computation happens, are efficient.

Let us illustrate with an example. The autocovariance (of lag $\ell$) of a sequence $x_1, \ldots, x_T$ is given by

$$
\gamma_\ell(x) = \frac{1}{T} \sum_{t=1}^{T-\ell} (x_t - \overline{x})(x_{t+\ell} - \overline{x})
$$
Thus we take $x$, put it in deviations from the mean, chop off the first $\ell$ elements to get, say, $x$-tail, chop off the last $\ell$ elements to get $x$-head, take the inner product of $x$-tail and $x$-head, and divide by $T$. Here it is in Lisp.

```lisp
(defun conv (lag x)
  (inner-product (butlast x lag) (butfirst x lag))
)
(defun butfirst (x &optional (n 1))
  (select x (which (<= n (iseq (length x)))))
)
```

If we want to compute the first $\text{maxlag}$ autocovariances, we use mapping and say

```lisp
(defun autocovar (x maxlag)
  (let ((n (length x))
        (z (- x (mean x))))
    (/ (mapcar #'(lambda (k) (conv k z)) (iseq maxlag)) n))
)
```

It seems that this should be quite efficient, because as we have seen most of the computation is done in $\text{blas-ddot}$, which is the engine of $\text{inner-product}$. So let's give it a try. The machine is a 300 MHz G3 with 128 MB of RAM, with MacOS 8.6, adn with XLISP-STAT 3.52.9 (beta). We apply it to the Zürich sunspot data, a series of length 2820, first by using raw Lisp.

```lisp
> (symbol-function 'autocovar)
#<Closure-AUTOCOVAR: #57eb358>
> (time (autocovara 100))
The evaluation took 4.47 seconds; 2.88 seconds in gc.
```

The large amount of garbage collecting indicates there is not enough memory available in the workspace. We say

```lisp
> (expand 100)
100
> (time (autocovara 100))
The evaluation took 2.20 seconds; 0.42 seconds in gc.
```

In this case, byte compiling does not make a difference.

```lisp
> (compile 'autocovar)
AUTOCOVAR
> (symbol-function 'autocovar)
#<Byte-Code-Closure-AUTOCOVAR: #57dbc58>
> (time (autocovardata 100))
The evaluation took 2.58 seconds; 0.68 seconds in gc.
```
In all cases, the computations seem to take about 1.6 – 1.8 seconds. This includes the mapping and the selection from the list.

It is well known, for instance Newton [1988, p. 24], that the autocovariance can be computed by applying two Fourier transforms to the sequence. This is implemented in the function below.

```lisp
(defun autocovar (x maxlag)
  (let* ((n (length x))
         (m (+ n maxlag))
         (y (append (- x (mean x)) (repeat 0 maxlag)))
         (r (/ (squared-abs (fft y)) m))
         (a (* (/ m n) (/ (realpart (fft r)) m))))
    (select a (1+ (iseq maxlag))))
)
(defun squared-abs (x)
  (let ((r (realpart x))
         (c (imagpart x)))
    (+ (* r r) (* c c))
)
```

We now find

```lisp
> (time (autocovar data 100))
The evaluation took 0.12 seconds; 0.02 seconds in gc.
```

This is a dramatic difference. It is due to the fact that now almost all of the computing is done on the C level, with virtually no manipulation of lists. Besides that, the subr `fft` implements the fast Fourier transform, which seems to be living up to its name.

3. **Shared Libraries: Example 1**

Let us discuss a simple first example of using shared libraries and the wrapper system. We share write a C version of the function `conv` we earlier did in Lisp. The file `cconv.c` looks like this.

```c
double cconv (int l, int n, double * x)
{
  double s = 0.0, * y = x + (n - l), * z = x + l, * u = x;
  while (u < y)
    s += *u++ * *z++;
  return (s);
}
```

Writing the additional glue code that links the C function to the XLISP-STAT application is the next step. This is OS dependent and not very simple. Tierney has written XLISP-STAT functions that writes these “wrappers” for you.
In our application the wrapper is in the file \texttt{conv.wrp} below.

\begin{verbatim}
(wrap:c-lines "double cconv (int, int, double *);")
(wrap:c-function base-conv "cconv"
  (:integer :integer (:cptr "double")) :flonum)

(defun conv (lag vec)
  (let ((n (length vec))
        (vec (coerce vec '(vector c-double))))
      (base-conv lag n
               (wrapptrs::cast-c-double (array-data-address vec)))))
\end{verbatim}

The first two lines are the critical ones, the second part is an \texttt{XLISP-STAT} function that calls the \texttt{C} function in the library. If we run \texttt{conv.wrp} through the make-wrappers function, using

\begin{verbatim}
(wrap:make-wrappers "conv.wrp")
\end{verbatim}

it generates a file \texttt{conv.lsp} and a file \texttt{conv.c}. The \texttt{conv.lsp} file has a byte-compiled version of the Lisp code in the wrapper file, but instructions to load the shared library. The \texttt{conv.c} file contains the glue to link the \texttt{C} function to the \texttt{XLISP-STAT} system. The \texttt{wrap:c-lines} and \texttt{wrap:c-function} are two macros in the \texttt{wrap} package that determine the structure of the \texttt{conv.c} file. Observe that \texttt{wrap:c-function} gives the Lisp symbol name given to the \texttt{C} function, and it explains the type of the arguments and the result. Also observe the handling of pointers in the \texttt{conv} Lisp function, using macros from the \texttt{wrapptrs} package. In order for everything to work you need to load the file \texttt{conv.lsp}, which creates the \texttt{wrap} package, and to load the file \texttt{wrapptrs.lsp}, which loads the shared library \texttt{wrapptrs.dll} and creates the package \texttt{wrapptrs}. The \texttt{wrapptrs} package itself is already and example of applying \texttt{wrap:make-wrappers} to the file \texttt{wrapptr.wrp}.

Now link \texttt{conv.c} and \texttt{cconv.c} into a shared library \texttt{conv.dll}. In the link you should also include the \texttt{XLISP-STAT} interpreter, and whatever libraries from your development system needed. Loading \texttt{conv.lsp} into the \texttt{XLISP-STAT} interpreter makes the shared library (and thus the \texttt{C} function) available. And here is the result.

\begin{verbatim}
> (time (autocovardata 100))
The evaluation took 0.43 seconds; 0.20 seconds in gc.
\end{verbatim}

We see an improvement compared to the raw Lisp with a factor of 5. Not as good as using the \texttt{fft}, but surprisingly good anyway. It seems that the Lisp \texttt{inner-product} function, which is a Lisp wrapper for \texttt{blas-ddot}, does introduce quite a bit of overhead.
4. Shared Libraries: Example 2

We can expect more gain in situations where the Lisp function we are replacing have many loops and list manipulations. In this example we take a function, written by Rick Schoenberg, to make contours on a scatterplot. XLISP-STAT has a `contour-function`, which draws the contours of a function of two variables. So in order to draw contours in a scatterplot, using \( n \) data-points \((x_i, y_i, z_i)\), we need a smoother that interpolates the function and the use `contour-function` on this smoother. Schoenberg uses a low-pass Gaussian two-dimensional filter to do the interpolations. Here is the Lisp code.

```lisp
(defun rsmooth-2d (i j x y z b1 b2)
  (let* ((w (/ (* (exp (/ (- (^ (- x i) 2)) (* 2 b1 b1)))
                    (exp (/ (- (^ (- y j) 2)) (* 2 b2 b2)))))
          (* 2 pi b1 b2)))
    (/ (sum (* z w)) (sum w)))
)

(defun rcontour (x y z
  &key levels (xnum 5) (ynum 5)
    (x1 (min x)) (x2 (max x)) (y1 (min y)) (y2 (max y))
    (b1 (/ (- x2 x1) (log (length z))))
    (b2 (/ (- y2 y1) (log (length z))))
    (smoother #'rsmooth-2d))
  (contour-function #'(lambda (i j)
    (funcall smoother i j x y z b1 b2))
    x1 x2 y1 y2 :num-points xnum :levels levels)
)
```

Clearly it is essential to make the smoother efficient. Here is a C version, in the file `cgauss.c`. 
#include <math.h>
#define pi 3.141592653589793
#define SQUARE(x) ((x) * (x))

double gaussian_2_smooth (int n, double x0, double y0,
    double b1, double b2, double * x, double * y, double * z) {
    int i; double sum1 = 0.0, sum2 = 0.0,
    term1, term2, term3, weight;
    for (i = 0 ; i < n ; i++) {
        term1 = exp (- SQUARE(x0 - *(x + i)) / (2.0 * SQUARE(b1)));
        term2 = exp (- SQUARE(y0 - *(y + i)) / (2.0 * SQUARE(b2)));
        term3 = 2.0 * pi * b1 * b2;
        weight = (term1 * term2) / term3;
        sum1 += weight * *(z + i);
        sum2 += weight;
    }
    return (sum1 / sum2);
}

The wrapper file gauss.wrp is

```
(wrap:c-lines "double gaussian_2_smooth
    (int, double, double, double, double *, double *, double *);")
(wrap:c-function rsmooth-2d-base "gaussian_2_smooth"
    (:cptr "double") (:cptr "double") (:cptr "double") :flonum)

(defun rsmooth-2d (i j x y z b1 b2)
    (let ((n (length x))
        (x (coerce x '(vector c-double)))
        (y (coerce y '(vector c-double)))
        (z (coerce z '(vector c-double))))
        (rsmooth-2d-base n i j b1 b2
            (wrapptrs:cast-c-double (array-data-address x))
            (wrapptrs:cast-c-double (array-data-address y))
            (wrapptrs:cast-c-double (array-data-address z))))
    )

(defun rcontour (x y z
    &key levels (xnum 5) (ynum 5)
    (x1 (min x)) (x2 (max x)) (y1 (min y)) (y2 (max y))
    (b1 (/ (- x2 x1) (log (length z))))
    (b2 (/ (- y2 y1) (log (length z))))
    (smoother #'rsmooth-2d))
    (contour-function #'(lambda (i j)
        (funcall smoother i j x y z b1 b2))
        x1 x2 y1 y2 :num-points xnum :levels levels) )
```
If we link `gauss.c` and `cgauss.c`, we must make sure that code for the `exp` function is also linked in, by using some sort of C math library. After loading `gauss.lsp` we have

```lisp
> (symbol-function 'rsmooth-2d-base)
#<Subr: #40b9888>
> (symbol-function 'rsmooth-2d)
#<Closure-RSMOOTH-2D: #40bbb18>
```

Now for the time comparison. The example

```lisp
(defvar x (* (uniform-rand 1000) 10))
(defvar y (* (uniform-rand 1000) 10))
(defvar z (+ (* (- x 3) (- x 3)) (* (- y 5) (- y 5))))
(rcontour x y z :levels (iseq 20))
```

takes 1.68 seconds on the G3 in the Lisp version, and 0.18 seconds in the C version. Almost ten times faster, and in many applications, possibly much larger than this example, that can make a huge difference.

### 5. Shared Libraries: Example 3

In this example, we use FORTRAN to replace C. Given the enormous amount of legacy numerical code in FORTRAN, this is an important extension. Obviously using FORTRAN presupposes we have a development system that compiles and links both languages. We shall use the Absoft compilers, running in the MPW environment.

In FORTRAN, we can write external functions and subroutines. External functions return a value, so they seem especially appropriate, but we shall look at using subroutines as well. One important property of FORTRAN is that arguments to subroutines or functions are passed by reference and not by value. This means that if we call FORTRAN routines from C, we pass pointers to the parameters, but in the FORTRAN routine itself we calculate as if values were passed. This makes life just a tiny bit more complicated.

Again, we will use the same convolution example. The FORTRAN source, in the file `fconv.f`, is

```fortran
real*8 function cconv (l, n, x)
integer*4 l, n
real*8 sum, x(n)
sum = 0.0
do 10, i=1,n - l
  sum=sum + x(i) * x(i + l)
cconv = sum
return
end
```
As a wrapper file we use

```
(wrap:c-lines "double cconv (int *, int *, double *);"
(wrap:c-function base-conv "cconv"
  ((:cptr "int") (:cptr "int") (:cptr "double") :flonum)
(defun conv (lag vec)
  (let ((n (coerce (list (length vec)) '(vector c-int)))
    (lag (coerce (list lag) '(vector c-int)))
    (vec (coerce vec '(vector c-double))))
    (base-conv (wrapptrs:cast-c-int (array-data-address lag))
      (wrapptrs:cast-c-int (array-data-address n))
      (wrapptrs:cast-c-double (array-data-address vec)))
  )
)
```

This works exactly the same as the C version. But observe that it needs the hack, where scalars are converted to one-element vectors (which we need to do in order to use `array-data-address`).

The same result can be attained by using a FORTRAN subroutine. The code is

```
subroutine cconv (l, n, x, sum)
   integer*4 l, n
   real*8 sum, x(n)
   n = 0.0
   do 10, i=1,n - l
     sum = sum + x(i) * x(i + l)
  10 return
end
```

and the wrapper code is

```
(wrap:c-lines "void cconv (int *, int *, double *, double *);"
(wrap:c-function base-conv "cconv"
  ((:cptr "int") (:cptr "int")
   (:cptr "double") (:cptr "double") :void)
(defun conv (lag vec)
  (let* ((n (coerce (list (length vec)) '(vector c-int)))
    (lag (coerce (list lag) '(vector c-int)))
    (vec (coerce vec '(vector c-double)))
    (sum (coerce (list 0.0) '(vector c-double))))
    (base-conv (wrapptrs:cast-c-int (array-data-address lag))
      (wrapptrs:cast-c-int (array-data-address n))
      (wrapptrs:cast-c-double (array-data-address vec))
      (wrapptrs:cast-c-double (array-data-address sum)))
    (aref (pointer-protected (array-data-address sum)) 0)
  ))
```
Appendix A. Introduction

In the Appendices we give documentation and references necessary to use the various ports to XLISP-STAT we have made so far. Generally, each of these libraries defines a package, and we give documentation for the external symbols.

It is best to set up the packages using the new autoload system Tierney [1998a].

The documentation below is incomplete, because documentation strings have not been added to all external functions yet. The appendices will be updated regularly if more documentation (and more modules) are added. We plan to add modules for generalized eigenvalue and singular value computation for nonsymmetric matrices, for writing pdf, for producing pdf and ps plots. We also plan to add functions to the optimization, solving, and smoothing modules. But this may take a long time.

If you are interested in obtaining the wrappers and libraries for these modules, just drop me an email. Of course you can only use the compiled versions if you have a PowerMac of some sort. But the wrapper files, the _autoidx.lsp index files, and the code for the libraries will make it possible to compile your own versions.
Appendix B. cephes

The cephes library is written, in C, by Stephen Mosier. The material is discussed extensively in the book Mosier [1989]. The source code is on netlib, at

www.netlib.org/cephes/index.html

We have not used all of cephes, only the special function part. The documentation below is copied in many cases from the source code.

1 DRAND:
2 Args: (&optional (n 1))
3 Returns a typed vector with n uniform random numbers between
4 1.0 and 2.0 using the Wichman-Hill generator.

5 ZETA:
6 Args: (x)
7 \[ zetac(x) = \sum_{k=2}^{\infty} \frac{1}{k}, \quad x > 1, \]
8 is related to the Riemann zeta function by
9 \[ \text{Riemann zeta}(x) = zetac(x) + 1. \]

10 PSI:
11 Args: (x)
12 \[ \psi(x) = \frac{d}{dx} \ln |x| \]
13 is the logarithmic derivative of the gamma function.

14 DAWSON:
15 Args: x
16 \[ dawsn(x) = \exp(-x) \int_0^x \exp(t^2) \, dt \]
INVERSE-INCOMPLETE-BETA:
Args: (a b y)
Given y, the function finds x such that
\[
x = \int_{(a+b)}^{(a-1 b-1)} \frac{t (1-t)}{(a-1) (b-1)} \, dt = y.
\]

INCOMPLETE-GAMMA:
Args: (a x &key (complement nil))
If complement is nil
\[
igam(a,x) = \frac{1}{(a-1)} \int_{(a)}^{x} e^{-t} \, dt.
\]
else
\[
igamc(a,x) = 1 - igam(a,x)
\]

FRESNEL:
Args (x)
\[
C(x) = \int_{(x)}^{(x)} \cos(\pi/2 t^2) \, dt,
\]

\[ S(x) = \int_{-\infty}^{x} \sin(\pi/2 t^2) \, dt. \]

Returns a typed vector with \((s \, c)\).

COMPLEMENTARY-ERROR-FUNCTION:
NIL

RECIPROCAL-GAMMA:
Args: \((x)\)
Returns one divided by the gamma function of the argument.

MODIFIED-BESSEL-THIRD-KIND:
Args: \((x &key \ (exp \ nil) \ (order \ 0))\)
Modified Bessel functions of the third kind, of order 0 or 1 or of integer order.
For the functions of order 0 or 1, one can choose to use exponential scaling.

BESSEL:
Args: \((x &key \ (order \ 0))\)
Bessel functions of order 0 or 1 or of integer or non-integer order.

SPENCE:
Args: \((x)\)
\[ \text{spence}(x) = - \int_{1}^{x} ----- \, dt \]
\[ \int_{t-1}^{1} \]

BESSEL-SECOND-KIND:
Args: \((x &key \ (order \ 0))\)
Bessel functions of the second kind of order 0 or 1 or of integer and non-integer order.
FAC:

 Args (x)

 Returns the factorial of (the integer) x.

GAUSS–HYPERGEOMETRIC–2F1:

 Args: (a b c x)

 \[
 \text{hyp2f1}(a, b, c, x) = \binom{a}{b} \binom{b}{c} x^c \frac{1}{(c+1) \cdots (c+k)} (k+1)!
 \]

 MODIFIED–BESSEL:

 Args: (x &key (exp nil) (order 0))

 Modified Bessel functions of order 0 or 1 or of non-integer order.

 For the functions of order 0 or 1, one can choose to
 use exponential scaling.

 INCOMPLETE–BETA:

 Args: (a b x)

 \[
 \frac{x}{(a+b)} (1-t) dt \quad \text{for} \quad 0 \leq t \leq 1
 \]

 COMPLETE–ELLIPTIC–FIRST–KIND:

 Args: (m)

 \[
 K(m) = \int_0^{\pi/2} \frac{1}{\sqrt{1 - m \sin t}} dt
 \]
ELLIPITC-FIRST-KIND:
Args: (phi m)

\[
\phi - \left\lbrack \int_0^{\phi_m} \frac{2}{\sqrt{1 - m \sin t}} \, dt \right\rbrack
\]

STRUVE:
Args: (v x)
Computes the Struve function Hv(x) of order v, argument x.

JACOBIAN-ELLIPTIC:
Args: (u m)
Evaluates the Jacobian elliptic functions sn(u|m), cn(u|m),
and dn(u|m) of parameter m between 0 and 1, and real
argument u. These functions are periodic, with quarter-period on the
real axis equal to the complete elliptic integral
ellpk(1.0-m). Relation to incomplete elliptic integral:
If \( u = \text{ellik}(\phi_m, m) \), then sn(u|m) = sin(\phi),
and cn(u|m) = cos(\phi). Phi is called the amplitude of u.
Returns a typed vector with (sn cn dn \phi).

COMPLETE-ELLIPTIC-SECOND-KIND:
Args: (m1)

\[
\pi/2 - \left\lbrack \int_0^{\pi/2} \frac{2}{\sqrt{1 - m \sin t}} \, dt \right\rbrack
\]

SINE-COSINE-INTEGRALS:
Args: (x &key (hyperbolic nil))
Approximates the integrals

x
USING C WRAPPERS IN XLISP-STAT

\[ \text{Ci}(x) = \text{eul} + \ln x + \int_{0}^{x} \frac{\cos t - 1}{t} \, dt, \]

where \( \text{eul} = 0.57721566490153286061 \) is Euler's constant. If \( \text{HYPERBOLIC} \) is \( t \), then hyperbolic sines and cosines are used. Returns results in a typed vector (c s).

**ERROR-FUNCTION:**

Args: \((x)\)

\[ \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} \exp(-t^2) \, dt. \]

**CONFLUENT-HYPERGEOMETRIC-1F1:**

Args: \((a \ b \ x)\)

\[ F(a,b;x) = 1 + \frac{a x}{1!} + \frac{a(a+1) x}{2!} + \ldots \]

**INVERSE-COMPLEMENTED-INCOMPLETE-GAMMA:**

Args: \((a \ p)\)

Given \( p \), the function finds \( x \) such that

\[ \int_{-t}^{\infty} \frac{1}{(a-1)!} \, dt = p \]
p = \int \frac{1}{e^{-t} t} \, dt.

\begin{align*}
p &= \frac{1}{e^{-t} t} \bigg|_{0}^{\infty} \\
&= 1 \\
&= \Gamma(a) - x
\end{align*}

\text{GAMMA:}
\text{Args: (x)}
\text{Returns the gamma function of x.}

\text{EXPONENTIAL-INTEGRAL:}
\text{Args: (n x)}
\begin{align*}
E(x) &= \int_{n}^{\infty} e^{-xt} \, dt \\
&= \frac{e^{-n}}{n} - \frac{e^{-1}}{1}
\end{align*}

\text{AIRY:}
\text{Args: x}
\text{Solves the differential equation } y''(x) = xy. \text{ The two independent solutions a and b, and their derivatives } a' \text{ and } b', \text{ at } x \text{ are returned in the typed vector (a a' b b'}).
Appendix C. GD

GDIMAGESETBRUSH:
Args: (gd gd_brush)
GD and GD_BRUSH are gdImagePtrs. The image in GD_BRUSH is
used as a brush in the image in GD. Returns NIL.

GDIMAGEGETINTERLACED:
Args: (gd)
GD is a gdImagePtr, the function returns T if the
image is interlaced and NIL if it is not.

GDIMAGETCOLORTRANSPARENT:
Args: (gd color)
Sets the index of the transparent color in the image
pointed to by GD to COLOR. If there are no transparent
colors, call this function with COLOR = -1. Returns NIL.

GDIMAGEPOLYGON:
Args: (gd x y color)
GD is a gdImagePtr, and COLOR is an integer corrsponding to
one of the colors of the image. X and Y are lists with the
coordinates of the vertices of the polygon. A polygon is drawn
in the image. Returns NIL.

GDIMAGEINTERLACE:
Args: (gd interlace)
If INTERLACE is T, the image pointed to by GD will be interlaced, if
INTERLACE is NIL it will not. Returns NIL.

GDIMAGECOLORDEALLOCATE:
Args: (gd color)
Deallocates the color indexed by COLOR in the image
pointed to by GD. Returns NIL.

GDIMAGEDASHEDLINE:
Args: (gd start_x start_y end_x end_y color)
Draws a dashed line from (START_X,START_Y) to (END_X,END_Y) in the image pointed
to by GD. Deprecated. Use gdImageSetStyle instead. Returns NIL.

GDIMAGECHAR:
Args: (gd gf x y char color)
GD is a gdImagePtr, and COLOR is an integer corrsponding to
one of the colors of the image. X and Y are the starting
coordinates of the character CHAR which is drawn horizontally
from left to right in size GF. Returns NIL.

GDIMAGECOLOREXACT:
Args: (gd r g b)
Returns the index of the allocated color in the image pointed to by GD that has RGB-values R, G, and B. Returns -1 if there is no such color.

GDIMAGECREATEFROMGIF:
Args: (filename)
Reads GIF file from FILENAME, and returns a gdImagePtr to an image. Returns NIL.

GDIMAGECOPYRESIZED:
Args: (gd_dst gd_src dst_x dst_y src_upper_left_x src_upper_left_y src_width src_height dst_width dst_height)
Copies and possibly resizes a rectangular region from the image pointed to by GD_SRC by GD_DST. The region copied has the upper left corner (SRC_UPPER_LEFT_X,SRC_UPPER_LEFT_Y) and width SRC_WIDTH and height SRC_HEIGHT. The region is copied to the point (DST_X,DST_Y) with width DST_WIDTH and height DST_HEIGHT. Returns NIL.

GDIMAGECOPY:
Args: (gd_dst gd_src dst_x dst_y src_upper_left_x src_upper_left_y width height)
Copies a rectangular region from the image pointed to by GD_SRC to the image pointed to by GD_DST. The region copied has the upper left corner (SRC_UPPER_LEFT_X,SRC_UPPER_LEFT_Y) and width WIDTH and height HEIGHT. The region is copied to the point (DST_X,DST_Y).

GDIMAGESETSTYLE:
Args: (gd style)
GD is a gdImagePtr and STYLE is a list of allocated colors of the image. Defines a style color for dashed lines. Returns NIL.

GDIMAGEFILL:
Args: (gd start_x start_y color)
Floods a portion of the image pointed to by GD with COLOR. The portion flooded is the surrounding region of the point (START_X,START_Y) with the same color as the starting point. Returns NIL.

GDIMAGEDESTROY:
Args: (gd)
Destroys the image pointed to by GD. Returns NIL.

GDIMAGECOLORSTOTAL:
Args: (gd)
GD is a gdImagePtr, the function returns the number of currently allocated colors in the image.
GDIMAGESTRING:
Args: (gd gf x y string color)
GD is a gdImagePtr, and COLOR is an integer corresponding to
one of the colors of the image. X and Y are the starting
coordinates of the STRING which is drawn horizontally
from left to right in characters of size GF. Returns NIL.

GDIMAGETGETPIXEL:
Args: (gd row col)
Returns the color value of the pixel in ROW and COL
of the image pointed to by GD.

GDIMAGECOLORALLOCATE:
Args: (gd r g b)
Allocates a color in the image pointed to by GD, with
RGB-values R, G, and B. Returns the color index.

GDIMAGEBLUE:
Args: (gd color)
GD is a gdImagePtr, and COLOR is one of its allocated colors.
The function returns the blue component of the color.

GDIMAGEGREEN:
Args: (gd color)
GD is a gdImagePtr, and COLOR is one of its allocated colors.
The function returns the green component of the color.

GDIMAGESETTILE:
Args: (gd gd_tile)
GD and GD_TILE are gdImagePtrs. The image in GD_TILE is
used as a tile in the image in GD. Returns NIL.

GDIMAGERECTANGLE:
Args: (gd upper_left_x upper_left_y lower_right_x lower_right_y color)
Draws a rectangle in color COLOR with upper left corner at (UPPER_LEFT_X,UPPER_LEFT_Y)
and lower right corner at (LOWER_RIGHT_X,LOWER_RIGHT_Y) in the image pointed to by

GDIMAGERED:
Args: (gd color)
GD is a gdImagePtr, and COLOR is one of its allocated colors.
The function returns the red component of the color.

GDIMAGEFILLTOBORDER:
Args: (gd start_x start_y color)
Floods a portion of the image pointed to by GD with FLOOD_COLOR. The portion flooded begins at the point (START_X,START_Y) and stops at border with color BORDER_COLOR. Returns NIL.

GDIMAGECOLORCLOSEST:
Args: (gd r g b)
Returns the index of the allocated color in the image pointed to by GD that is closest to the color with RGB-values R, G, and B.

GDIMAGECREATE:
Args: (nrow ncol)
Returns a gdImagePtr to an (empty) image with a height of NROW pixels and a width of NCOL pixels.

GDIMAGEGD:
Args: (gd filename)
GD is a gdImagePtr. The corresponding image is written in GD format to the file FILENAME. Returns NIL.

GDIMAGEARC:
Args: (gd start_x start_y end_x end_y color)
Draws a segment of an ellips in color COLOR centered at (CENTER_X,CENTER_Y), of width WIDTH and height HEIGHT, starting at BEGIN_DEGREE and ending at END_DEGREE in the image pointed to by GD. Returns NIL.

GDIMAGELINE:
Args: (gd start_x start_y end_x end_y color)
Draws a line from (START_X,START_Y) to (END_X,END_Y) in the image pointed to by GD. Returns NIL.

GDIMAGESX:
Args: (gd)
GD is a gdImagePtr, the function returns the width of the image in pixels.

GDIMAGESY:
Args: (gd)
GD is a gdImagePtr, the function returns the height of the image in pixels.

GDIMAGESSTRINGUP:
Args: (gd gf x y string color)
GD is a gdImagePtr, and COLOR is an integer corresponding to one of the colors of the image. X and Y are the starting
coordinates of the STRING which is drawn vertically
from bottom to top in characters of size GF. Returns NIL.

GDIMAGEGETTRANSPARENT:
Args: (gd)
GD is a gdImagePtr, the function returns the current
transparent color of the image.

GDIMAGESETPIXEL:
Args: (gd row col)
Sets the color value of the pixel in ROW and COL
of the image pointed to by GD. Returns NIL.

GDIMAGECREATEFROMGD:
Args: (filename)
Reads GD file from FILENAME, and returns
a gdImagePtr to an image.

GDIMAGECHARUP:
Args: (gd gf x y char color)
GD is a gdImagePtr, and COLOR is an integer corresponding to
one of the colors of the image. X and Y are the starting
coordinates of the character CHAR which is drawn vertically
from bottom to top in size GF. Returns NIL.

GDIMAGEGIF:
Args: (gd filename)
GD is a gdImagePtr. The corresponding image is written
in GIF format to the file FILENAME. Returns NIL.

GDIMAGEFILLEDRECTANGLE:
Args: (gd upper_left_x upper_left_y lower_right_x lower_right_y color)
Draws a filled rectangle in color COLOR with upper left corner at (UPPER_LEFT_X,UPPER_LEFT_Y) and lower right corner at (LOWER_RIGHT_X,LOWER_RIGHT_Y) in the image pointed to by GD.

GDIMAGECREATEFROMXBM:
Args: (filename)
Reads XBM file from FILENAME, and returns
a gdImagePtr to an image.

GDIMAGEFILLEDPOLYGON:
Args: (gd x y color)
GD is a gdImagePtr, and COLOR is an integer corresponding to
one of the colors of the image. X and Y are lists with the
coordinates of the vertices of the polygon. A filled polygon
is drawn in the image. Returns NIL.
APPENDIX D. pppack

1 REINSCH-SMOOTHING-SPLINE:
2 Args: (x y &key (dy (repeat 1.0 (length x))) (s (float (length x))))
3
4 PIECEWISE-POLYNOMIAL-VALUE:
5 NIL
6
7 PIECEWISE-POLYNOMIAL-FROM-B-SPLINE:
8 NIL
9
10 ALL-B-SPLINE-VALUES:
11 Args: (knot jhigh x left)
12 Computes the values of all non-zero B-splines with knots KNOT of order JHIGH at X.
13 Here LEFT is the index such that KNOT[LEFT] < X < KNOT[LEFT+1], and KNOT has
14 length LEFT + JHIGH. We must have JHIGH <= 20.
15
16 CUBIC-SPLINE-INTERPOLANT-VALUE:
17 NIL
18
19 B-SPLINE-VALUE:
20 NIL
21
22 CUBIC-SPLINE-INTERPOLANT:
23 Args: (x y jderiv &key (plot t) (min (min x)) (max (max x)) (numpoints 100))
24 Plots the JDERIV-th derivative of the cubic interpolating spline through the scat
25 sequences x and y.
26
27 B-SPLINE:
28 Args: (knot bcoef jderiv &key (min (min knot)) (max (max knot)) (numpoints 100))
29 Plots the JDERIV-th derivative of the B-spline with (n + k) knots KNOT and n coeff
30 Note: k = length (KNOT) - length (BCOEF) is the order of the spline. KNOT is suppo
31 be nondecreasing.
The specfun library is another collection of special functions, which has a great deal of overlap with cephes. The code was written, in FORTRAN, by W.J. Cody. The source code is on netlib, at

www.netlib.org/specfun/index.html

There is also a version in

www.netlib.org/toms/715

which corresponds with Cody [1993].

Of special interest, perhaps, is the function machar, which dynamically computes machine parameters Cody [1988]. If called from XLISP-STAT, we obtain

ibeta = 2
it = 53
irnd = 5
ngrd = 0
machep = -52
negeps = -53
iexp = 11
minexp = -1022
maxexp = 1024
eps = 2.220446049250313e-16
epsneg = 1.1102230246251565e-16
xmin = 2.2250738585072014e-308
xmax = 1.7976931348623157e+308

1  BESY0:
2  NIL
3
4  BESY1:
5  NIL
6
7  RIBESL:
8  NIL
9
10  EXPEI:
11  NIL
12
13  BESJ0:
14  NIL
15
16  DLGAMA:
17  NIL
18
19  BESJ1:
20  NIL
21
22  BESK0:
23  NIL
24
25  BESK1:
26  NIL
27
28  BESI0:
29  NIL
30
31  PSI:
32  NIL
33
34  BESI1:
35  NIL
36
37  DERFC:
38  NIL
39
40  DAW:
41  NIL
42
43  DGAMMA:
44  NIL
45
46  MACHAR:
47  NIL
48
49  BESEK0:
50  NIL
51
52  BESEK1:
53  NIL
54
55  BESEI0:
56  NIL
57
58  BESEI1:
59  NIL
60
61  RYBESL:
62  NIL
63
64  RJBESL:
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>NIL</td>
</tr>
<tr>
<td>66</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>RKBESL:</td>
</tr>
<tr>
<td>68</td>
<td>NIL</td>
</tr>
<tr>
<td>69</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>EONE:</td>
</tr>
<tr>
<td>71</td>
<td>NIL</td>
</tr>
<tr>
<td>72</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>ANORM:</td>
</tr>
<tr>
<td>74</td>
<td>NIL</td>
</tr>
<tr>
<td>75</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>DERF:</td>
</tr>
<tr>
<td>77</td>
<td>NIL</td>
</tr>
<tr>
<td>78</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>DERFCX:</td>
</tr>
<tr>
<td>80</td>
<td>NIL</td>
</tr>
<tr>
<td>81</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>EI:</td>
</tr>
<tr>
<td>83</td>
<td>NIL</td>
</tr>
</tbody>
</table>
Appendix F. Probability

NONCENTRAL-CHISQ-CDF
Args: (x dfr pnonc)
Returns the value of the Noncentral ChiSquare (DFR,PNONC) distribution function at X.

NONCENTRAL-CHISQ-QUANT
Args: (p dfr pnonc)
Returns the P-th quantile of the Noncentral ChiSquare (DFR,PNONC) distribution.

NONCENTRAL-F-CDF
Args: (x dfr1 dfr2 pnonc)
Returns the value of the Noncentral F (DFR1, DFR2, PNONC) distribution function at X.

NONCENTRAL-F-QUANT
Args: (p dfr1 dfr2 pnonc)
Returns the P-th quantile of the Noncentral F (DFR1, DFR2, PNONC) distribution.

NONCENTRAL-T-CDF
Args: (x dfr pnonc)
Returns the value of the Noncentral t (DFR,PNONC) distribution function at X.

NONCENTRAL-T-QUANT
Args: (p dfr pnonc)
Returns the P-th quantile of the Noncentral t (DFR,PNONC) distribution.
APPENDIX G. SMOOTHING

CUBIC-SPLINE-DATA-SMOOTHER:

Args: (x y &key (d (repeat 1.0 (length x))) (var -1.0) (job 0) (plot t))

Interface to Hutchinson's cubgcv cubic spline smoother. X has N abscissae,
Y has N ordinates. D are the relative standard deviations. If unknown, set
D = 1.0. If known, then set VAR = 1. If VAR < 0 then generalized cross-validation
is used to estimate the smoothing parameter, and VAR returns the error variance.
If VAR > 0 then the smoothing parameter is estimated by estimating the MSE and
VAR is unchanged. If VAR = 0 an interpolating cubic spline is calculated.
If JOB = 0 standard errors are not computed, if JOB =1 they are computed.
If PLOT is non-zero, the resulting smoother is plotted.

LOCAL-POLYNOMIAL-RIDGE-REGRESSION:
NIL

KERNEL-REGRESSION-LOCAL-BANDWIDTH:
NIL

KERNEL-REGRESSION-GLOBAL-BANDWIDTH:
NIL
APPENDIX H. SOLVING

1 CPOLY
2 Args: (coefs)
3 Computes the roots of a polynomial with complex coefficients COEFS.
APPENDIX I. OPTIMIZATION

QUADRATIC-PROGRAM

Args: (dmat dvec amat bvec meq &key (ierr 0))

This routine uses the Goldfarb/Idnani algorithm to solve the following minimization problem:

\[
\begin{align*}
\text{minimize} & \quad -d^T x + \frac{1}{2} \times^T D x \\
\text{where} & \quad A1^T x = b1 \\
& \quad A2^T x \geq b2
\end{align*}
\]

the matrix D is in DMAT, assumed to be symmetric and positive definite and of order n. Vector d is in DVEC. Matrix A, containing both A1 and A2 is in the n x q array AMAT, b is in the q-vector BVEC. MEQ indicates how many of the q constraints are equality constraints. The program returns a list with the optimum value, the number of iterations, the optimum solution, and the vector indicating which constraints are active.

FORTRAN by Berwin A. Turlach <bturlach@stats.adelaide.edu.au>.
Appendix J. transform

1  FCT2D:  
2   NIL   
3  
4  IFCT2D:  
5    NIL   
6  
7   FCT:  
8    NIL   
9  
10  IFCT: 
11    NIL
Appendix K. Density

LSCV-KD-CRITERION:
Args: (data bw)
DATA is some 1-d sequence, BW is the bandwidth.
Computes the least squares cross validation criterion
to be minimized in bandwidth selection for kernel density estimation

KERNEL-DENS-PLUGIN:
Args: (x &key (z (rseq (min x) (max x) 50)) (plot t) (kerndens t))

SHEATHER-JONES-SEQ-BANDWIDTH:
Args: (data)
Return sheather-jones solve-the-equation bandwidth for kernel density estimation
To be used with gaussian kernel. Data size must be less than 1600.

WARPED-HISTOGRAM:
Args: (x h m &key (k 3) (plot t))
X is a sequence of data. H is the bandwidth, and M is the number of
small bins in a large bin. K is the kernel (1: uniform, 2: triangle,
3: epanechnikov, 4: quartic, 5:triweight, and PLOT indicates if a plot
should be made. KERNDENS adds a dashed kernel-density plot.
REFERENCES


UCLA STATISTICS DEPARTMENT

E-mail address: deleeuw@stat.ucla.edu